(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
18 October 2001 (18.10.2001)

PCT

(10) International Publication Number
WO 01/77935 A2

(51) International Patent Classification[7]: G06F 17/60

(21) International Application Number: PCT/US00/21289

(22) International Filing Date: 3 August 2000 (03.08.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/195,381 7 April 2000 (07.04.2000) US

(71) Applicant *(for all designated States except US)*: **MYPRIMETIME, INC.** [US/US]; 410 Jessie Street, Suite 300, San Francisco, CA 94103 (US).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **ELIAS, Samir** [US/US]; 112 Haviture Way, Hercules, CA 94547 (US).

(74) Agent: **MARTIN, Alice, O.**; 2600 Chase Plaza, 10 South LaSalle Street, Chicago, IL 60603 (US).

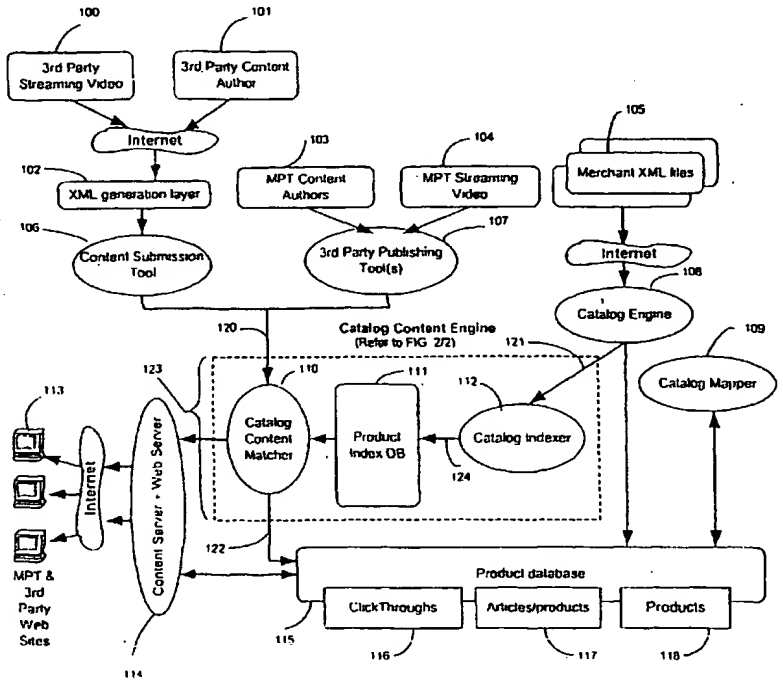
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: CONTEXTUAL E-COMMERCE ENGINE



(57) Abstract: The invention provides a method for integrating textual content documents created by authors with stores of information on relevant products and services and providing the integrated information to users for goal-oriented interactions. The integrating is useful in e-commerce, in particular for use in the Internet.

CONTEXTUAL E-COMMERCE ENGINE

Inventor: Samir Elias

BACKGROUND OF THE INVENTION

The invention provides a method for integrating textual content of documents created by authors with stores of information on relevant products and services, and presenting the integrated information to users for their goal-oriented interactions. The integrating is useful for accessing information across data networks such as the Internet and Broadband Telecommunications networks and is especially useful in e-commerce.

Many Web servers operate "Web sites" which offer a collection of lined hypertext documents controlled by a single person or entity. Because the Web site is controlled by a single person or entity, the hypertext documents, often called "Web pages" in this context, generally have a consistent look and subject matter. Especially in the case of Web sites put up by commercial interests selling goods and services, the hyperlinked documents which form a Web site will have few, if any, links to pages not controlled by the commercial interests. The terms "Web site" and "Web page" are often used interchangeably, but herein a "Web page" refers to a single hypertext document which forms part of a Web site and "Web site" refers to a collection of one or more Web pages which are controlled (i.e., modifiable) by a single entity (e.g. commercial interest) or group of entities working in concert to present a site on a particular topic.

Internet portals and online magazines provide interactive forums in which documents covering various topics are accessed by individuals through a computer. An application of the interactive nature of online portals and magazines is to provide links (Internet connections) to other Web sites that have products and services that are relevant to the articles displayed. However, creating links from articles to relevant products and services can be a laborious and time-consuming task.

Authors of documents may determine key phrases and words that characterize their documents and then they or users of the Internet may search databases with the

key words to find products and services that relate to the subject of the documents. Databases (or search engines) typically list query returns in order of relevance, weighted by the number of times the key words appear on the first page of a Web site. However, searches done on public databases and search engines can result in improperly weighted queries due to "spamming". Spamming occurs when a Web page author includes a word multiple times of times on a Web site merely to increase the weight it is assigned when searches are conducted using that particular key word. This procedure demands more effort from persons who need to examine the results of database searches to determine which Web sites should be linked to the document that was written. Regular search engines do not "learn" nor "remember" which web pages are cheating using spamming techniques. Through human feedback and storage of prior search results, our invention corrects for spamming. Authors or computer programmers then create links to Web sites that have products or services most relevant to a document.

Because a public database or search engine searches (results) are not usually saved in the memory of the database, new searches must be conducted each time links need to be found for a document. Because this is inefficient, there is a need for data storage, organization, and ranking methods that allow for more efficient searches and use of links to sites or other computer files. There is also a need for a feedback system in which the returned data from a search is ranked to effect more accurate data searches in the future.

SUMMARY OF THE INVENTION

The invention provides a contextual e-commerce engine that integrates the content of documents created by authors, with data on products/services and presents the integrated information to users for goal-oriented interaction. The invention is useful in e-commerce, in particular the Internet. The authors are generally experts in a field related to the content of the documents. The product/services data includes that provided by merchants who partner in Web sites that provide the documents, or by third parties who are not a formal part of the Web site.

The invention relates a method for automating interactions between textual content of documents and stores of information on relevant products and services.

The method includes searching an article for key words and phrases, searching the stored information for a match to the key words of the articles, mapping (linking) the results of the search between the articles and database, and merging the textual content of the article with an information file from the products and services database. The search is conducted so that the relevant weight of each information file is updated after each use.

The invention also provides a method for integrating product data from sources outside a Web site (server) into a catalog database used by the Web site. The method includes providing access to the database for the sources outside the Web site, receiving product data from the sources outside the Web site, storing the data in a persistent store, indexing the stored product data by key stored words, and mapping the indexed product data onto the categories of the Web site catalog database.

Definitions

API - Application Program Interface. When a library of code is created for use with more than one program, the developers usually expose a list of routines known as the API for accessing the library's functions.

Articles - Used interchangeably with **documents** herein to mean information provided by authors for example, on a Web page.

Broadband - A type of data transmission in which a single medium can carry several channels at once.

Categories - The product database is organized in part by placing the products into predefined categories. These categories are hierarchical and form a general ontology on the product database. Examples include: Computer products, printers, digital cameras.

Contextual Documents - See articles.

Flat file - There are many ways of storing data which is composed of a series of records of identical or similar structure. The simplest way is to store them in a file on disk, in sequential order. No relational structure between the records is stored.

Frequency relevance factor (FRF) - Each lexeme in the dictionary has associated with it a frequency of appearance in the entire body of product descriptions. Each lexeme appearing in a document being used as a query also has a frequency of

appearance within the document itself. The frequency relevance factor is calculated as a function of the two frequencies, and represents how relevant a particular lexeme is to finding matches for a particular document (article).

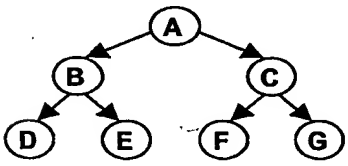
Inverted index - The inverted index extends the idea of an inverted list. A normal list of documents would simply have the contents of the documents, one after another. In contrast, an inverted list is a list of words in which each word has a list of documents it appears in.

Learned relevance factor (LRF) - As the system accumulates feedback responses, a scalar factor for each lexeme is adjusted to reflect how relevant it was in matching products to documents (articles). This factor is a real number, between –1.0 and +1.0 at all times.

Lexeme - When text is submitted to the catalog content engine, it is usually a character stream of fully punctuated and formatted English prose, and must be processed into a form that is usable by the engine. This process includes parsing the text into normalized-case tokens, removing the stopwords, and then stemming the tokens. After this processing, the words in stemmed form are called lexemes.

Metadata - Metadata is information about data; e.g. if the data is an article's text, metadata would include the byline, date and time of composition, and the like.

Persistent storage – Data that needs to be kept for long periods of time are stored in a way that survives major system maintenance procedures and most transient system failures. This usually means that the data is stored in files on a hard disk.



Prefix order - A tree data structure can be parsed in several ways that uniquely order the nodes. Prefix order means that each node is parsed before its children, sibling nodes are parsed left to right, and children are parsed before siblings. Sentinel indicators show that a node has no children. For given example tree, the prefix order (using a period as the sentinel character) would be: ABD..E..CF..G.. The original tree structure can be completely recovered from this parsing.

Skip-list - Skip-lists are an extension of linked lists, and they probabilistically improve search time to log(n) where **n** is the number of nodes in the list. Skip-lists are described extensively in Pugh (1990).

Skip-list index - Indices, by their nature, provide fast access to a collection of information, and so skip lists are a logical choice for implementation. A skip-list index is an index where the records are the elements in the list.

SKU – Stock Keeping Unit. A merchant specific unique identifier for a product.

Sparse matrix – A sparse matrix is a typically very large matrix with a high percentage of zero entries. Compression techniques known to those of skill in the art are applied to dramatically reduce the storage size and processing time on these matrices.

Stemming - Stemming is a process that converts words with multiple forms into one unique form. For example, the verb "drink" in the English language is lexically modified according to tense, and may appear in text as "drink", "drinking", or "drank." After stemming, all occurrences of the verb will appear as "drink."

Stopword removal - Certain words appear in English text so often that they are useless when attempting to distinguish one document from another. Stopword removal is the process of identifying and removing these words from processed text, so that only useful and relevant words remain. Typical stopwords include "in", "the", "is", "that", and so forth.

Third Party - Refers to a source outside of the control of an entity operating the primary Web site.

Token - A string of characters not containing white space, or a double-quote delimited string of any characters.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a flow chart illustrating the overall scheme of the Contextual E-commerce Engine™ Technology (CE²).™ There are different sources of data used in this system, including merchant product XML files 105 that describe products available from the merchant, documents/stories written by authors and submitted through a particular entity in control of a Web site (MPT) 103 or a 3rd party publishing system 101, and streaming video 100, 104 created by specialists and submitted

through a 3rd party publishing system or a particular entity.

FIG. 2 is a flow chart illustrating the components of the Catalog Content Engine (CCE) and is a more detailed depiction of the Catalog Content Engine area outlined in FIG. 1. The CCE submission server 200, implements the catalog content product matcher and is divided into three components: the submission server, which manages remote submission of content and feedback, the system manager 202, which provides an interface for basic system management and maintenance functions, and the CCE services API 201, which provides services for the processing and management of product information and article documents.

FIG. 3A and 3B illustrate a typical main page for a Web site using the Catalog Content Engine to display subject text and related products.

FIG. 4A and 4B illustrate a typical subject text and related products page for a Web site using the Catalog Content Engine.

FIG. 5A and 5B illustrate a typical merchant Web site from which a consumer can order products associated with subject text from the CCE using Web site.

DETAILED DESCRIPTION OF THE INVENTION

The invention is directed to a method that integrates textual content written by authors for a primary Web site, with data from products and services supplied by external sources, and provides the integrated information to users for goal-oriented interactions.

The invention provides a method for automating interactions between textual/streamed content of articles and database of relevant products and services, the method including searching the article for key words and phrases, searching the product database for a match to the key words of the articles, mapping (linking) the results of the search between the articles and database, and merging the textual/streamed content of the article with selected products/services. The search is conducted such that the relevant weight of each selected product/service entry is updated after each use.

The invention also provides a method for integrating product data from sources outside a Web site(server) into a catalog database used by the Web site, the method includes providing access to the database for the sources outside the Web site,

receiving product data from the source outside the Web site, storing the data in a product database, indexing the stored product data by key words, and mapping the indexed product data onto the categories of the Web site catalog database.

System Architecture:

There are different sources of data used in this system (refer to FIG. 1):

- Merchant product XML files that describe products available from the merchant 105.
- Articles/Stories written by authors 103, and submitted through a 3rd party publishing system 107.
- Articles/Stories written by other authors 101, that are not directly a part of the primary Web site and submitted through a content submission tool 106.
- Streaming video created by specialists 104, and submitted through a 3rd party publishing system 107.
- Streaming video created by other specialists 100, and submitted through a content submission tool 106.

The merchant product XML files are submitted through a web interface to a waiting CatalogEngine™ 108, which will extract the product information from the files and insert this information into the product database. The extracted product information will also be indexed into a fast access, incrementally updateable text retrieval database(Product Index DB 111) for later access by the CatalogContentMatcher™ 110. The product information is further refined by mapping the product categories to internally defined product categories using the CatalogMapper™ 109.

The article/story/video data is submitted by the content submission/publishing tools to the CatalogContentEngine™ 123, which will map(tie) the data to the products from the Product Index DB. The mapping is saved in the Article/Products database 117, for later access by the ContentServer™ 114, which serves the article/category/product relationship in a predefined XML/HTML format.

The invention is directed to a method that integrates textual/streamed content written by authors for a primary Web site, with products supplied by external sources. The external product and service sources use a web interface to download their

product catalogs written in XML. XML(Extended Markup Language) is a markup computer language. Used in this case to describe merchant products/services. Each product catalog has merchant name/category/brand/model/price/description/ image url/home url/status fields for each product/service.

The CatalogEngine™ 108, extracts the merchant name/category/brand/ model/description for each product/service from the XML file and passes it to the CatalogIndexer™ 112, and passes this information and all other product information to the Product Database 115.

The CatalogEngine™ 108, also creates a unique identifier for each product, this identifier is used to relate each product entry in the Product Database 115, to the product entry in the Product Index DB 111.

For each new group of product categories identified by the CatalogEngine™ 108, as categories new/different from what is in the Product Database 115, a notification is sent by the CatalogEngine™ 108, to a person responsible for mapping the new product categories to the internally defined categories.

The CatalogMapper™ 109, is used for mapping the identified product categories to product categories as specified(to a large extent) by the UN/Standard Product and Services Code(UNSPSC), an international, evolving standard. The mapping is kept together with product information in the Product Database 115.

The external product and service sources use a Web interface to download their product catalogs written to a specific format, generally in XML (extended markup language) to the user Web site. XML is a computer programing language that allows specific information to be electronically labeled as specific items such as a product name, cost, or size. The catalogs are retrieved by an internal database, which gathers all product and service information and stores it in a persistent file. Each product catalog has merchant name/category/brand/model/price/description/image url/home url/description for each product/service from the XML file and passes it to the CatalogIndexer™ and passes this information and all other product information to the Product Database. The CatalogEngine™ also creates a unique identifier for each product which is used to relate each product entry in the Product Database to the product entry in the Product Index Database. For each new group of product

categories identified by the CatalogEngine™ as categories new/different from what is in the Product Database, a notifications sent by the CatalogEngine™ to a person responsible for mapping the new product categories to the internal defined categories.

Catalog Mapper

Contextual Intelligent matching technology includes a tool used to map product and service categories from outside sources, to the private database categories. This function serves to place information files about a merchant product or service into appropriate categories of the primary Web site database. These categories are to a large extent based on the *UN/Standard Product and Services Code* (UNSPSC), an international, evolving, standard. The mapped categories are stored in the persistent file together with the product information.

Catalog Indexer

The indexed database categorizes products using Natural Language Processing to choose appropriate terms and phrases from product descriptions. This allows the accessing of products using the indexed terms and phrases. The product index database generated is a dynamically updateable database. New products/services can be indexed incrementally, so that there is no need to fully index the product database for each new merchant catalog.

The authors of the articles use third party publishing software known to those of skill in the art to write their stories/articles. The authors can hook up directly to a search engine which goes through an article/story and also scans the product index database looking for matching products/services. Upon finding a match, the search engine creates an article to product mapping in the persistent store. The search engine learns from writer feedback to make future choices more accurate.

Catalog Server

The local server creates a link by merging the article/story HTML text together with the categories/product HTML. Consumers (users) are then able to view the articles/stories along side the relevant categories and products. In addition to the users being able to view a listing of the private database catalog, they can also click through to the merchant partner product pages. A merchant partner is joined contractually to the proprietor of the Web site.

Catalog Context Engine

The Catalog Content Engine (CCE) is separated into two logical subsystems, each of which can be located on the same physical machine.

The Catalog Content Engine (CCE) 123 implements both the CatalogIndexer™ 112, and the CatalogContentMatcher™ 110, and has four components: (1) the submission server 200, which manages remote submission of content and feedback; (2) the system manager 202, which provides an interface for basic system management and maintenance functions (startup, shutdown, parameter modification); (3) the CCE services API 201, which provides services for the processing and management of product information and article documents; and (4) the Product Index DB 111, which provides storage services for managing the stored and processed information and metadata about articles and products. The system manager can be implemented in a variety of ways. The Product Index DB 111, provides persistent storage services for all of the components described above. These services are described below, in detail, in the **Serialization** section. The API is broken down into several kinds of entities:

A **product** object has its relevant information stored in several forms:

- the **inverted index**
- a list of keywords
- the vendor name
- the brand name
- a list of category names
- a list of noun-phrases

The **inverted index** contains the processed text of the product description, and is managed separately as a collection of **postings**. The inverted index is described herein. The product object contains all other fields, such as price. A particular product is uniquely defined by the vendor name, brand name, and model name/number, and is referable in the database by either a key constructed from those three fields, or the SKU/part number if available. The text of the product description is stored separately, in flat files.

The **categories** are stored in a tree structure, wherein the tree structure

matches the ontological structure of the categories. The root node of the tree does not correspond to any category, but is rather a container for all the top-level category nodes. Each node has a link to its parent node and its child nodes. The category tree also has a skip-list index to all the nodes, so that particular category nodes can be found quickly.

The **inverted index** is the core of the product Index DB. It is separated into two components, the dictionary and the postings. The dictionary is a list of all the lexemes known to the system, with a few essential statistics on each lexeme available. For each lexeme in the dictionary, there is a posting that stores which documents contain this lexeme, and with what frequency.

The **dictionary** itself is a skip-list of lexeme objects, where the key is the lexeme string itself. Each lexeme is the unique stemmed word form of a particular word. (Stemming is described herein, along with product document management.) Each lexeme includes: a count of the number of documents it appears in, the total number of appearances in the entire set of product documents, a frequency relevance factor (FRF) a learned relevance factor (LRF) that will be used in query processing, and a reference to the posting cache where the posting can be found.

A **posting** is an indexed sparse vector (i.e., a linked list) of floating point values, where each value is the number of appearances of the particular lexeme in each document, each indexed entry corresponds to a particular document, and the correspondence holds across all of the postings. The conjunction of all the posting vectors forms a matrix, and queries are executed by applying operations to this matrix, so postings should support basic sparse matrix operations. Postings are stored in several posting caches, according to size, as described herein.

A **posting cache** is a limited-size collection of posting objects employing the standard least-recently-used algorithm for determining which postings to keep in memory. There is a list of caches, one for each scale class of posting. A scale class caching system is a scheme in which several different caches are made, according to posting size, where size is measured in the number of non-zero entries. For example, one cache which may hold all postings under 100 entries, another may hold postings under 200 entries, a third may hold posting under 400 entries, and so forth. The best

size and number of caches to use are determined empirically, by examining the distribution of posting sizes itself. This scheme can be tuned to maintain continuous retention of any particular class of postings.

An **article** (document) object stores the processed information corresponding to an article, including a list of lexemes, a list of keywords, lists of vendor names, product names, and categories found in the text, a list of noun-phrases, and a unique key to index the article in the database. The Product Index Database provides persistent storage services for all of the components described above. These services are described herein in detail in the **Serialization** section.

The **Catalog Content Engine** also supports **Catalog Indexing.** When a product description is submitted for catalog indexing, it includes a vendor name, a brand name, a model name/number, a list of keywords, a list of categories, and a product description. This is typically presented as a character stream of standard English prose, fully punctuated and formatted, but must be processed for use in the system. The punctuation is removed, the letter case normalized, and the text parsed into a stream of tokens. The token stream is used for several purposes. To obtain the list of lexemes for indexing, **stopword removal** is performed, which eliminates frequently appearing words. **Stemming** is then applied to convert differing forms of the same word into one form, and then convert the results to a list of unique tokens appearing in the document, keeping track of how many times each lexeme appears. The original token stream is then scanned for noun-phrases, which are subsequently stored on a list. To add the product description to the inverted list, either an unused column in the posting matrix is chosen or a new column is created if no unused columns exist. For each lexeme in the list, the count of lexeme elements in this product description is placed into the posting in the corresponding matrix column. If new lexemes are encountered, a new dictionary entry and posting are created for the new lexeme. All other product information is stored in indexed random access text files.

The Catalog Content Engine also supports **Catalog Content Matching.** When an article is submitted to the Catalog Content Matcher for matching to products, it is a character stream of standard English prose text, fully punctuated and

formatted, accompanied by a list of highly relevant keywords. The text must be processed for use in the system. In the same manner as the product descriptions, the punctuation is removed, the letter case is normalized, and the text is parsed into a stream of tokens. This stream of tokens is used for several separate product-matching tasks. To conduct the initial query, the token stream is converted to a list of unique lexemes and lexeme counts as described for product descriptions. These lexemes select certain rows out of the whole matrix. A sub-matrix that contains only those rows is constructed for performing operations on it. The frequency relevance factor for a particular lexeme is calculated in the following way (Byoung-Gyu):

$$FRF(lex) = \log(1/\text{corpusAppearanceCount}(lex)) \times \log(\text{articleAppearanceCount}(lex))$$

lex : the lexeme in question

corpusAppearanceCount(*lex*) : how many times *lex* appears in all product descriptions

articleAppearanceCount(*lex*) : how many times *lex* appears in the article text

FRF(*lex*) : the frequency relevance factor for *lex*.

The frequency relevance factor is calculated for one lexeme at a time, and then the log sum of the current row plus the frequency plus the learned relevance factor (adjusted for product appearance count) is accumulated with the previously processed rows, dividing the results by the total number of rows to normalize the sums when complete, as follows (Byoung-Gyu):

$$\text{Accum}(prod) \leftarrow \text{Accum}(prod) + FRF(lex) + LRF(lex) \times \log(\text{productAppearanceCount}(prod, lex))$$

prod : the product in question

productAppearanceCount(*prod, lex*) : how many times *lex* appears in the description of the product *prod*

Accum(*prod*) : the accumulated relevance for the product *prod* so far

After performing this process for each lexeme, only one row of values exists, in which each value represents a level of confidence in a particular product's relevance to the query. A set of the top-ranking products is then selected and used in further processing.

After the top ranking product is set, the keyword and noun-phrase matching measures are applied to further distinguish them. Each measure produces a separate score for the document, which is then combined in a weighted sum. A weighted sum is a repeated addition in which each added term is first multiplied by a weighting term. The keyword, vendor name, product name, and category names can all be matched by passing through the token stream and checking for matches against the top-ranking products, keeping counts of all matches per product. Then calculate relevance factors for each measure as follows (Vogt, 1996):

$\text{rel}(matchClass, product) = \log(\text{matchCount}(matchClass, product))$

matchClass : the type of keyword under consideration (categories, vendor names, and the like)

$\text{rel}(matchClass, product)$: the relevance factor for a particular product under a particular keyword type

"Match class" refers to which particular item is counted: keywords, vendor and product names, or category names. Noun-phrases are parsed out of the token stream, and then the list of noun-phrases is compared to the list of noun-phrases for each product and matches are counted and processed as discussed above.

After a separate set of relevance scores for each measure and product has been determined, the scores are combined in a weighted sum as follows:

$$\text{rel}(product) = \sum_{matchClasses} \text{relFactor}(matchClass) \times \text{rel}(matchClass, product)$$

product : the product in question

$\text{relFactor}(matchClass)$: our belief in the predictive ability of this keyword type

The predictive ability is useful because in this implementation, each individual matching technique (here called a match class: keyword, category, vendor names) is given an individual weight representative of its fitness in producing good matches.

To produce a better match, the score from each technique is multiplied by the weight, so that good techniques affect the score more than inferior techniques. Using a combination of techniques is better than relying on a single technique, because it greatly reduces the difference between the match score and what might be an ideal match score. In statistical parlance, it smooths out noise. This process yields a single relevance score for each product in the top-scoring set of products.

Serialization

The **Catalog Content Engine** also features a persistent storage serialization mechanism, implemented through the posting cache system described earlier. Each cache has a corresponding directory on disk in which the posting vectors from the inverted list index are stored, sorted in subdirectories according to the first two characters of the lexeme. When products are added to the inverted list index, the Catalog Content Engine updates the appropriate posting vectors in each cache, and then the caches in turn manage serializing the updated information to disk. When products are deleted from the inverted list index, the place they occupy is marked with a "tombstone" (an industry standard file processing technique) so that the data they contain will not be considered in query processing, and so that the system manager can reclaim the space later. When a large number of tombstones have accumulated, the system manager can reclaim the space by physically removing the corresponding data from each vector in the inverted list index. This procedure is very resource intensive, as it requires modifying every part of the entire index, and should therefore be performed offline.

Each **posting vector** from the inverted list stores the lexeme itself, the frequency count of lexeme occurrences in the entire set of product descriptions, the frequency relevance factor, the learned relevance factor, and the frequency count of the lexeme for each product in whose information the lexeme appears.

The **posting files** for each cache are stored in a subdirectory system consisting of one directory for each letter of the alphabet, each of which contains the posting files for all lexemes beginning with that letter.

The **category tree** is stored in a single file. The tree is written to the file in prefix order, with a special indicator marking the end of a list of sub-branches of a particular branch in the tree. Prefix order dictates that each branch in the tree is written before its sub-branches.

The **persistent information** that needs to be serialized includes the dictionary, the inverted index postings, the products metadata, and the category tree, as well as the index structures on each. Each of these groups of information is stored in a separate folder on the disk, to make the organizational structure clear. Each file has a header indicating what purpose the file serves.

The **dictionary** is stored as a series of lexeme entries. Each entry includes the lexeme itself, the count of occurrences in the entire set of product descriptions, the frequency relevance factor, the learned relevance factor, and the filename of the posting, which is derivable from the lexeme itself.

A **posting** from the inverted index is stored in a file named after the lexeme, and the posting files are stored in a directory system consisting of one directory for each letter of the alphabet, each of which contains the posting files for all lexemes beginning with that letter. An individual posting file contains the lexeme and the floating point posting values.

The **metadata** on the products is stored in a binary file as a series of product entries. The first part of the file indicates how many products are stored in the file. Each product entry begins with the product's ID, followed by the vendor name, the brand name, and the model name/number. Following that is a list of keywords, a list of category names, a count of the number of noun-phrases found in the product description, and then the noun-phrases themselves.

The recommended implementation is for each file indexed with multiple records stored within it to have an index file built. Each entry in the index file corresponds to one record, and has two things in it: a key to the record, and an integer indicating the record's offset position within the file, in bytes. When a record is added to the main file, the file's size is noted, and then the record is appended. The index can then be updated by adding an entry with the record's key, and the offset is the same as the file's previous size.

Utility services

A number of utility services are needed by the system in multiple places. These include efficient fast-access containers (skip-lists), fast string matching routines, and text parsing routines. Skip lists are described in detail by Pugh. Text parsing is straightforward, and standard tokenizing routines can be used.

Feedback learning

The system supports its own modification through feedback, so that it can be tuned for more effective matching. The facilitating mechanism is a set of tunable learned relevance factors (LRFs) for lexemes in the index, which combine with the frequency relevance factors to give more accurate weights for each lexeme's relevance. Initially, these relevance factors are all uniform to render their effects invisible. As feedback indicates which lexemes are most helpful in matching products to articles, the LRFs are modified accordingly.

Feedback is obtained by presenting a match of an article and product to a human expert and asking whether this match is acceptable, expecting a reply of either "yes" or "no." This is the sole feedback component that the system requests. From this it is determined whether the lexemes that contributed heavily to this match were useful in determining the match. If the expert rejects the match, this new knowledge is encoded by adjusting the LRFs for the heavily contributing lexemes downward by a small amount. If the expert approves the match, the LRFs is adjusted for the heavily contributing lexemes upward by a small amount. At all times, the LRFs are greater than 0.0 and less than 1.0. The actual adjustment are found by counting total positive and negative reinforcements, and then calculating as follows [Vogt]:

LRF(*lex*) = (**arctan**(totalPositives(*lex*) – totalNegatives(*lex*)) + 1) / 2

totalPositives(*lex*) : total number of positive reinforcements for the lexeme *lex*

total negatives(*lex*) : total number of negative reinforcements for the lexeme *lex*

LRF(*lex*) : the learned relevance frequency for the lexeme *lex*

Over a period of time, the LRFs of the highly useful lexemes will tend toward 1.0, increasing their influence, and the LRFs of the less useful lexemes will tend toward 0.0.

EXAMPLES

The following examples illustrate embodiments of the invention.

EXAMPLE 1: MYPRIMETIME™ WEB SITE ARTICLE SCREENS

The primary interface experienced by users of the Contextual E-Commerce Engine involves a series of Web sites or pages designed to display internally created articles that match relevant products and services from outside merchants. Web site consumer interfaces on a particular Web site the MyPrimeTime™ (MPT) Web sites (FIGS. 3A and B, 4A and B, 5A and B) serve as an illustrative, step-wise example of how a consumer may navigate the article screens of a system utilizing the Contextual E-Commerce Engine.

EXAMPLE 2: SUBMITTING A PRODUCT

The typical situations for using the CCE Services Protocol Description (see Materials and Methods) fall into two categories: article related and product related. The product related situations are fairly straightforward, and are illustrated by this example.

An illustration of how submitting a new product's information proceeds is as follows:

Source	Message
Client	**COMMAND: SUBMIT** **OPERAND: PRODUCT** **ID: SKU1123823723** **VENDOR: Egghead** **BRAND: Dragon Systems** **MODEL: DRAGON DICTATE CLASSIC ED V3.0 CD** **CATEGORY: Software|Business Productivity** **TEXT: Fast and accurate speech recognition system that instantly converts the spoken word into written text on the PC screen. Classic Edition has 120,000 word dictionary w/30,000 words in RAM.** **Ideal for users who create documents on**

	Variety of subjects.
Server	**100 OK**

EXAMPLE 3: SUBMITTING AN ARTICLE

The typical situations for using the CCE Services Protocol Description (see Materials and Methods) fall into two categories: article related and product related. The product related situations are fairly straight forward, and are illustrated by example 2. Article related situations are somewhat more complex because they involve communication with a human: the author. In general, the author follows the same steps for each session: write the article; submit the article for a test query; process feedback; adjust some parameters for matching; submit the article for storage. The other two scenarios illustrate these interactions.

An outline of the events that take place when an article (document) is submitted for indexing. Here, the article in TEST mode, which indicates that the client will request detailed information on matches and provide feedback.

Source	Message
Client	**COMMAND: SUBMIT** **OPERAND: ARTICLE** **MODE: TEST** **ID: ART00107** **TITLE: Designs on Sound** **KEYWORDS: audio|sound** **TEXT: Myprimetime speaks with sound editor Dane Davis, whose work on The Matrix snagged an Academy Award nomination.**
Server	**400 Session: S0011** **100 OK**

EXAMPLE 4: FEEDBACK DURING ARTICLE SUBMISSION

The typical situations for using the CCE Services Protocol Description (see Materials and Methods) fall into two categories: article related and product related. The product related situations are fairly straight forward, and are illustrated by the

first scenario. Article related situations are somewhat more complex because they involve communication with a human: the author. In general, the author follows the same steps for each session: write the article; submit the article for a test query; process feedback; adjust some parameters for matching; submit the article for storage. This example illustrates these interactions.

Example outlining how a feedback request and response cycle proceeds, after submitting an article.

Source	Message
Client	**SESSION: S0011** **COMMAND: MATCH** **HIGH: 0.8** **LOW: 0.5** **RESULT: TOP** **COUNT: 5** **GROUP: FIRST**
Server	**400 Session: S0011** **406 Count: 12** **405 Match: SKU1123823723 0.941** **405 Match: SKU1243897123 0.934** **405 Match: SKU1373127323 0.919** **405 Match: SKU1476342167 0.912** **405 Match: SKU1541783427 0.897**
Client	**SESSION: S0011** **COMMAND: FBRESPONSE** **DECLINE: SKU1243897123** **DECLINE: SKU1541783427**
Server	**400 Session: S0011** **100 OK**
Client	**SESSION: S0011** **COMMAND: MATCH** **HIGH: 0.8** **LOW: 0.5** **RESULT: TOP** **COUNT: 5**

	GROUP: NEXT
Server	**400 Session: S0011** **405 Match: SKU0123823723 0.875** **405 Match: SKU0243897123 0.845** **405 Match: SKU0373127323 0.832** **405 Match: SKU0476342167 0.811** **405 Match: SKU0541783427 0.808**
Client	**SESSION: S0011** **COMMAND: EXPLAIN** **ID: SKU0373127323** **COUNT: 5**
Server	**400 Session: S0011** **402 Lexeme: imaging 0.8501** **402 Lexeme: technique 0.754** **402 Lexeme: voice 0.711** **402 Lexeme: computer 0.601** **402 Lexeme: key 0.4123** **403 Keyword: high-tech** **404 Category: computer**
Client	**SESSION: S0011** **COMMAND: FBRESPONSE** **ACCEPT: SKU0123823723** **DECLINE: SKU0243897123** **DECLINE: SKU0373127323** **DECLINE: SKU0476342167** **ACCEPT: SKU0541783427**
Server	**400 Session: S0011** **100 OK**
Client	**SESSION: S0011** **COMMAND: ENDSESSION**
Server	**100 OK**

MATERIALS AND METHODS

Catalog Content Engine Protocol Description

The CatalogContentEngine™ 123, provides access to the catalog indexing and content matching services via the Catalog Content Engine Protocol (CCEP) described below.

This client-server protocol is text-based, explicitly session-based, connectionless, client-side command driven, and can be implemented for remote access on TCP. In particular, the server accepts new articles and products for indexing, processes requests for deleting articles and products from the index, executes matching queries, gathers feedback data, and performs maintenance functions on itself. The client chooses which commands to execute, sends command requests, and the server responds as necessary.

The commands that can be executed follow a general syntax: command name on the first line, command operand on the next line, followed by zero or more parameters on subsequent lines. This structure is called a **command block**. The character set for commands is US-ASCII. The standard white-space characters (space, horizontal tab, vertical tab) are the token delimiters. Certain command parameters are parsed verbatim, as required. Command names and parameters are not case sensitive, except where case-sensitivity is required for some reason (i.e. IDs, article text, and the like). Command blocks are terminated by two sequential new-line characters. Every command block issued will elicit at least one response message from the server indicating the result of the command's attempted execution, and sometimes multiple response messages will be sent.

Sessions are maintained by the explicit request of the **client.** When a client issues a command block, if no session ID is given, the server will generate one if it may be necessary. Certain commands do not establish a session, and therefore will not cause a session ID to be generated. If a client wishes to continue a session, the given session ID should be specified before the command block, as illustrated in the examples below. Clients should indicate when they no longer require a session causing the session to be closed. The server will invalidate all sessions which are idle beyond a specified time limit.

Command parameters are specified in standard regular expression syntax. Italics indicate a variable that can be any token meaningful to the command. A token

is a string of characters not containing white space, or a double-quote delimited string of any characters. Special cases include *any*, which indicates that any (possibly empty) string of characters is acceptable and taken verbatim in most cases, and *none*, which indicates that no parameters should be specified. The following is a list of the commands that can be processed by the server:

(Note that parameters preceded by [1] may appear only once, those preceded by [m] may appear more than once, and those preceded by * are required.)

Submission of an Article

Command:	SUBMIT
Operand:	ARTICLE
Parameters:	[1] MODE: (TEST|STORE)
	[1] ID: *articleID*
	[1] TITLE: *title*
	[m] KEYWORD: *keyword*{|*keyword*}
	[m] TEXT: *articleText*
Description:	Submit an article as a query to the catalog content engine. If the MODE TEST parameter is specified, the server processes the article without permanently storing it, and results are only available during this session. If the parameter MODE STORE is specified, the server processes and stores the article. Note that if an article ID is not specified, one will be generated and returned by the server if the MODE STORE parameter is specified.
Responses:	100 OK
	301 Missing parameter
	302 Parameter syntax error
	410 ID Generated: *articleID*

Submission of a Product

Command:	SUBMIT
Operand:	PRODUCT
Parameters:	[1] ID: *productID*
	[1] VENDOR: *vendorName*
	[1] BRAND: *brandName*

[1] MODEL: *modelName*

[1] CATEGORY: *categoryName*(|*categoryName*)

[m] TEXT: *productDescriptionText*

Description: Submit an article as a query to the catalog indexer. The META parameter allows the client to indicate various fields that are not of direct interest to the indexer, such as a URL for purchase, etc. Note that if a product ID is not specified, one will be generated and returned by the server. The generated ID will be formed by the concatenation of the vendor name, the brand name and the model name.

Responses: 100 OK

301 Missing parameter

302 Parameter syntax error

410 ID Generated: *productID*

Command name: FBRESPONSE
Operand: *None*
Parameters: [m] ACCEPT: *productID*

[m] DECLINE: *productID*

Description: Return a set of feedback responses to the server regarding the most recently submitted article. If the match is acceptable, the ACCEPT parameter is used, and if the match is not acceptable, the DECLINE parameter is used.

Responses: 100 OK

301 Missing parameter

302 Parameter syntax error

Command name: EXPLAIN
Operand: *None*
Parameters: [1*] ID: *productID*

[1*] COUNT: *maxLexemes*

Description: Given a product ID, explain the match with the most recently submitted article by giving the most relevant lexemes and the

matching keywords, categories, brands, and vendors.

Responses: 302 Parameter syntax error

402 Lexeme: *lexeme relevance*

403 Keyword: *keyword*

404 Category: *category*

Command name: MATCH

Operand: *None*

Parameters: [1] ID: *articleID*

[1*] HIGH: *highCutoff*

[1*] LOW: *lowCutoff*

[1*] RANKING: (top|mid|low)

[1*] COUNT: *count*

[1*] GROUP: (first|next)

Description: Request product matches for the most recently submitted article. These matches are the ones actually selected for the Web site. If an article ID is specified, retrieves this article and treats it as the most recently submitted. The server applies the high and low cutoff values to determine in which result category a match belongs.

Responses: 301 Missing parameter

304 Parameter value error

306 Invalid article ID

401 Product: *productID*

Command name: ENDSESSION

Operand: *None*

Parameters: *None*

Description: Indicate that a session is no longer needed and should be closed.

Responses: 100 OK

Server response messages fall into several classes: server status information, error messages, and data transmission to the client. Each class has a range of numbers assigned to it, by hundreds: 100 for status information, 300 for error messages, and 400 for data transmission. Server response messages are always given in the same

format: Message number, followed by a space, the message text, and a new-line character. The following is a list of possible server response messages:

Number	Text	Description
100	OK	The command was successfully executed.
101	Unknown error	The command was unsuccessfully executed, for some unknown reason.
102	Internal server error	The server encountered some internal error while processing a command.
301	Missing parameter	The command expected one or more parameters that were not found.
302	Parameter syntax error	The parameters could not be understood.
303	Invalid command	The first token found is not a command.
304	Parameter value error	The parameter values were out of the allowed range.
305	Ignored parameter	An unneeded parameter was found and ignored.
306	Invalid article ID	An invalid article ID was given.
307	Invalid session ID	An invalid session ID (one either not issued by the server or one that has been closed) was given as part of a command block.
400	Session: *sessionID*	The server issued a new session ID, or used the indicated ID for the transaction.
401	Product: *productID*	A product ID returned as the result of a command (e.g. MATCH).
402	Lexeme: *lexeme relevance*	A lexeme and its numeric relevance to a particular match, returned as the result of an EXPLAIN command.

403	Keyword: *keyword*	A keyword returned as the result of an EXPLAIN command.
404	Category: *category*	A category returned as the result of an EXPLAIN command.
405	Match: *productID relevance*	An article-product matching pair returned as the result of a FBREQUEST command.
406	Count: *count*	An integer indicating how many article-product matches are available for feedback, returned as the result of a MATCH command.
410	ID Generated: *itemID*	If a product or article is submitted without an ID, the server generates one and returns it in this message.

DOCUMENTS CITED

Brown, Eric W., James P. Callan and W. Bruce Croft. Fast incremental indexing for full-text information retrieval. Proceedings of the 20th VLDB conference; 1994.

Byoung-Gyu, Chang. Survey on the implementation of IR system, inverted file and the ranking algorithm. Survey paper; Korea Advanced Institute of Science and Technology, Unknown Date.

Campbell, Malcolm. An evaluation of indexing techniques for text-based information retrieval systems. MS thesis; Chisholm Institute of Technology; November 1989.

Knight, Jon P. and Martin Hamilton. A file system based inverted index. Technical report: LUT CS-TR 996; Loughborough University of Technology dept. of computer studies; 1996.

Kolda, Tamara G., and Dianne P. O'Leary. Latent semantic indexing via a semi-discrete matrix decomposition.

Letsche, Todd A. and Michael W. Barry. Large-scale information retrieval with latent semantic indexing. Information Sciences-Applications; 1996.

Pugh, William. Skip Lists: A probabilistic alternative to balanced trees. Communications of the ACM; June 1990.

Tomasic, Anthony, Hector Garcia-Molina, and Kurt Shoens. Incremental updates of inverted lists for text document retrieval. Technical note: STAN-CS-TN-93-1; Stanford University computer science dept.; December 1993.

Vogt, Christopher C., Garrison W. Cottrell, Richard K. Belew, and Brian T. Bartell. Using relevance to train a linear mixture of experts. Proceedings of the Text Retrieval Conference; 1996.

Zobel, Justin, Alistair Moffat, and Ron Sacks-Davis. An efficient indexing technique for full-text database systems. Proceedings of the 18th VLDB conference; 1992.

WE CLAIM:

1. A means for retreiving information that integrates the context of documents with relevant product and services data and means for presenting the integrated information to users.

2. A method for dynamically creating a primary Web page by merging the textual content of documents with information files on relevant products and services, said method comprising:

(a) searching the documents for key words and phrases;

(b) searching the information files for at least one match to the key words of the documents;

(c) merging the textual content of the documents with the information files on the products and services.

3. The method of claim 2, wherein the relevant weight of each information file is updated.

4. A method for mapping product data obtained from sources outside a local server into a catalog database used to dynamically create the primary Web site, said method comprising the steps of :

(a) obtaining access to the sources outside the Web site;

(b) receiving product data from the sources outside the Web site;

(c) storing the data in a persistent store;

(d) indexing the stored product data by key stored words; and

(e) mapping the indexed product data onto the categories of the Web site catalog database.

5. The method of claim 4 further comprising sending and receiving data between a remote and local computer in command blocks using a predetermined protocol.

6. The method of claim 5, wherein said protocol comprises:

(a) a command block syntax wherein each command block includes a command name on the first line, a command operand on the next line, at least zero parameters on subsequent lines, and is terminated by two sequential new line characters;

(b) a command set wherein the command set is compliant with the US-ASCII standard;

(c) the command block syntax which is not case sensitive; and

(d) standard white-space characters wherein the standard white-space characters are token delimiters.

7. An apparatus for facilitating the integration of the context of documents with product and services data comprising:

a storage device; and

a processor connected to the storage device,

the storage device storing

a program for controlling the processor documents, and product services data; and

the processor operative with the program to integrate textual content of documents with information files on relevant products and services.

8. An apparatus of claim 7, in which the processor is further operative with the program to update the relevant weight of each information file.

9. An apparatus of claim 8, in which the processor is further operative with the program to integrate product data from sources outside a local server into a catalog database used to dynamically create the primary Web site.

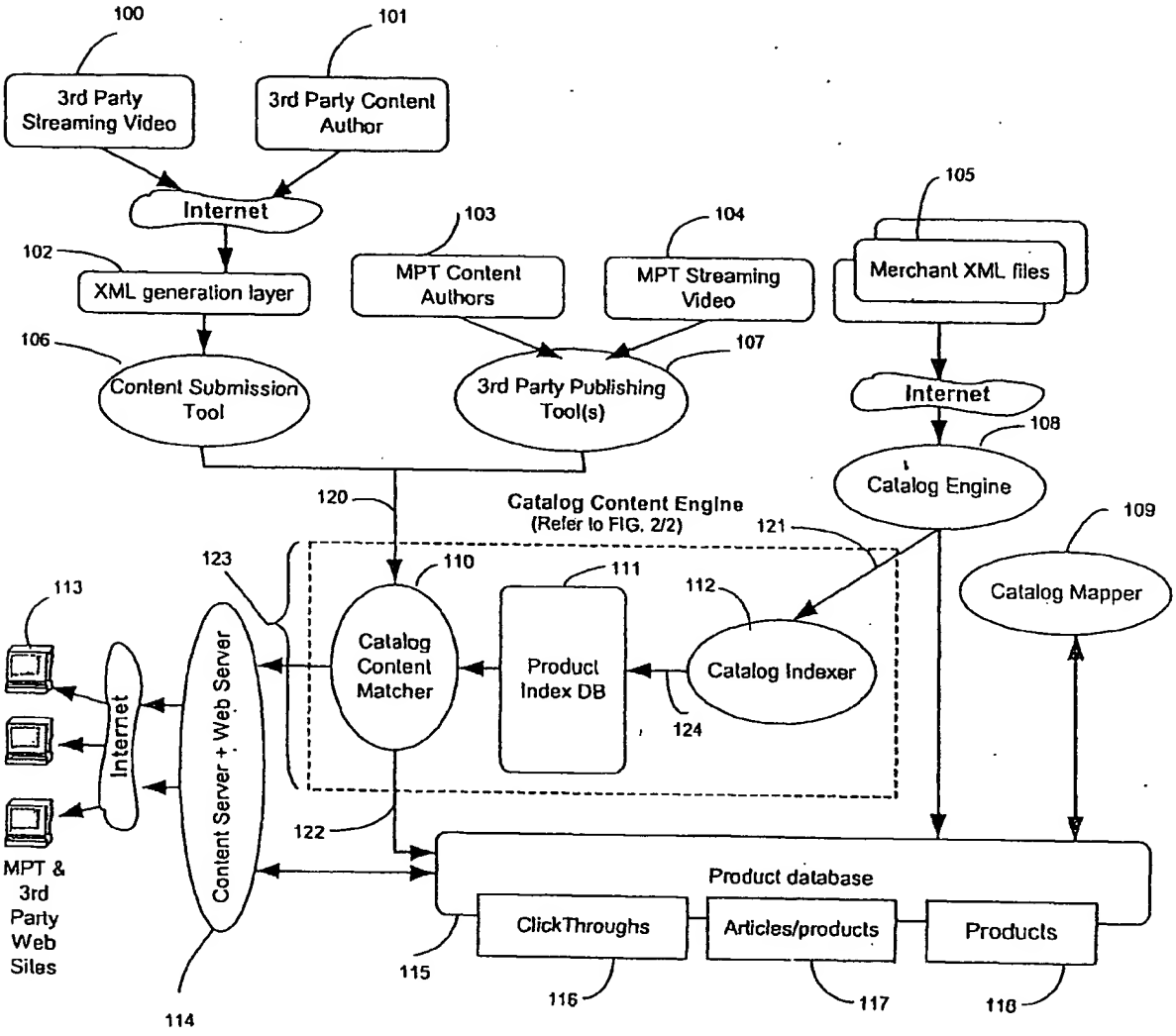

100
101
3rd Party Streaming Video
3rd Party Content Author
Internet
102
XML generation layer
103
MPT Content Authors
104
MPT Streaming Video
105
Merchant XML files
106
Content Submission Tool
3rd Party Publishing Tool(s)
107
Internet
108
Catalog Engine
120
Catalog Content Engine
(Refer to FIG. 2/2)
121
109
Catalog Mapper
123
110
111
112
113
Catalog Content Matcher
Product Index DB
Catalog Indexer
124
Internet
Content Server + Web Server
122
Product database
ClickThroughs
Articles/products
Products
MPT & 3rd Party Web Sites
115
116
117
118
114

FIG. 1

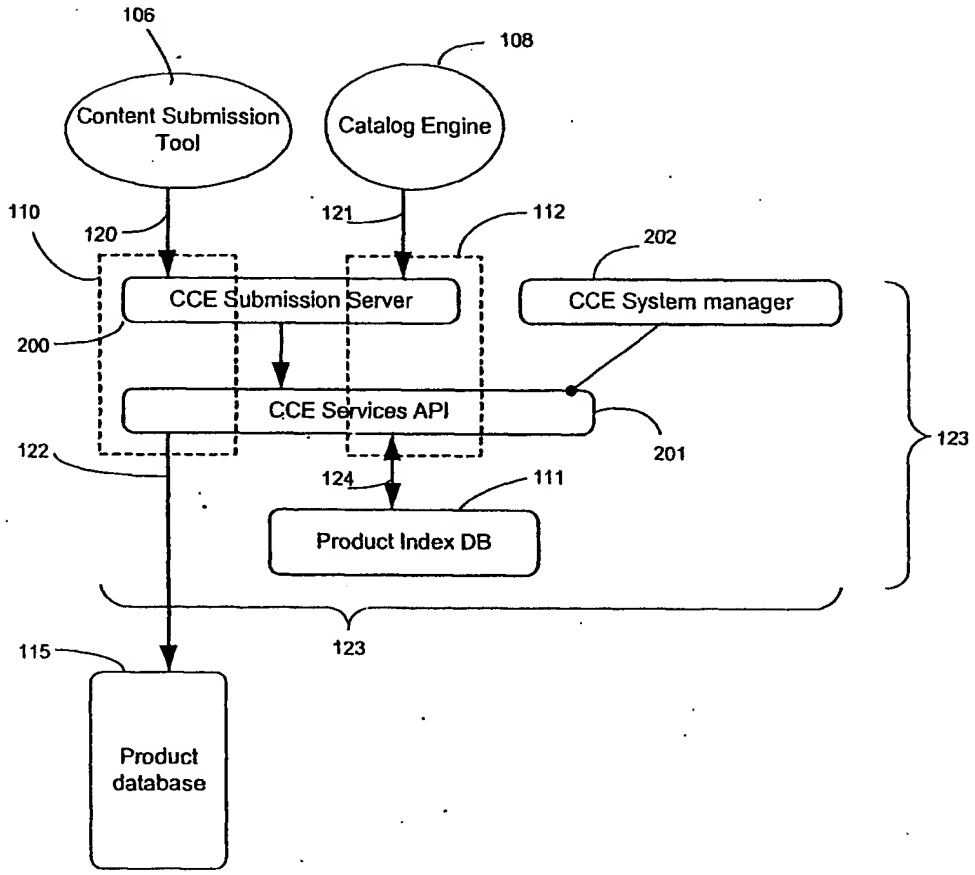
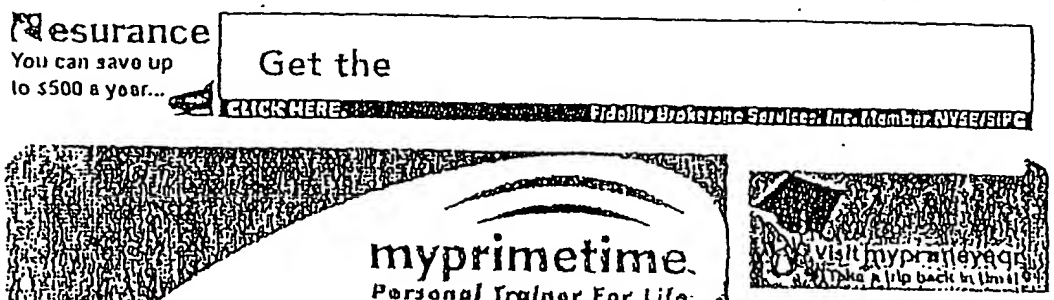

106
108
Content Submission Tool
Catalog Engine
110
120
121
112
202
CCE Submission Server
CCE System manager
200
CCE Services API
201
123
122
124
111
Product Index DB
123
115
Product database

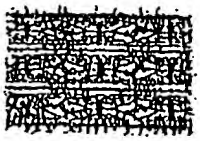
FIG. 2

MyPrimeTime: a baby boomers web portal offers tips on money, work, health and play



Today's Road Map:

- Stay Informed: Today's News, Weather and Stocks
- Do You Truly Love Your Job? Take the Work Quiz
- Find Out Where Women Are Slam Dunking Success
- No Time to Train? Get in Shape With Our Virtual Workout



Is Your Home Decor Right for You?

Take our quiz to learn what colors, shapes and styles provide all the comforts at home. (full story)

Ailments & Answers

Kids, Computers and Carpal Tunnel

Kids today will spend 23 years of their lifetime surfing the Web. Prevent them from having future back and wrist problems — check out our interactive guide to ergonomics. (full story)

Work & Life

Let Your Home Computers Work Together

You want to do Internet research, but your kid's downloading MP3s. With the simple and affordable networking kits available today, you can both share one Internet connection. (full story)

Real Estate

Safe as Houses

More Americans are plowing some of their stock profits into real estate. (full story)

FIG. 3A

MyPrimeTime: a baby boomers web portal offers tips on money, work, health and play

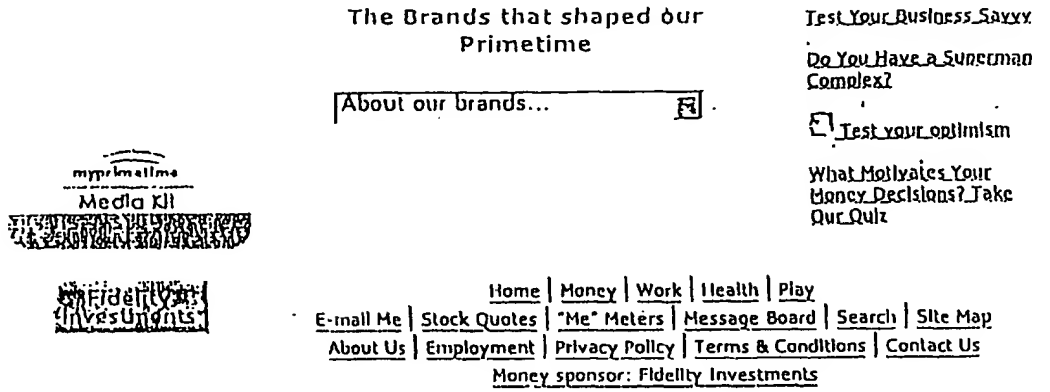


FIG 3B

Networking your home computers is getting easier and cheaper

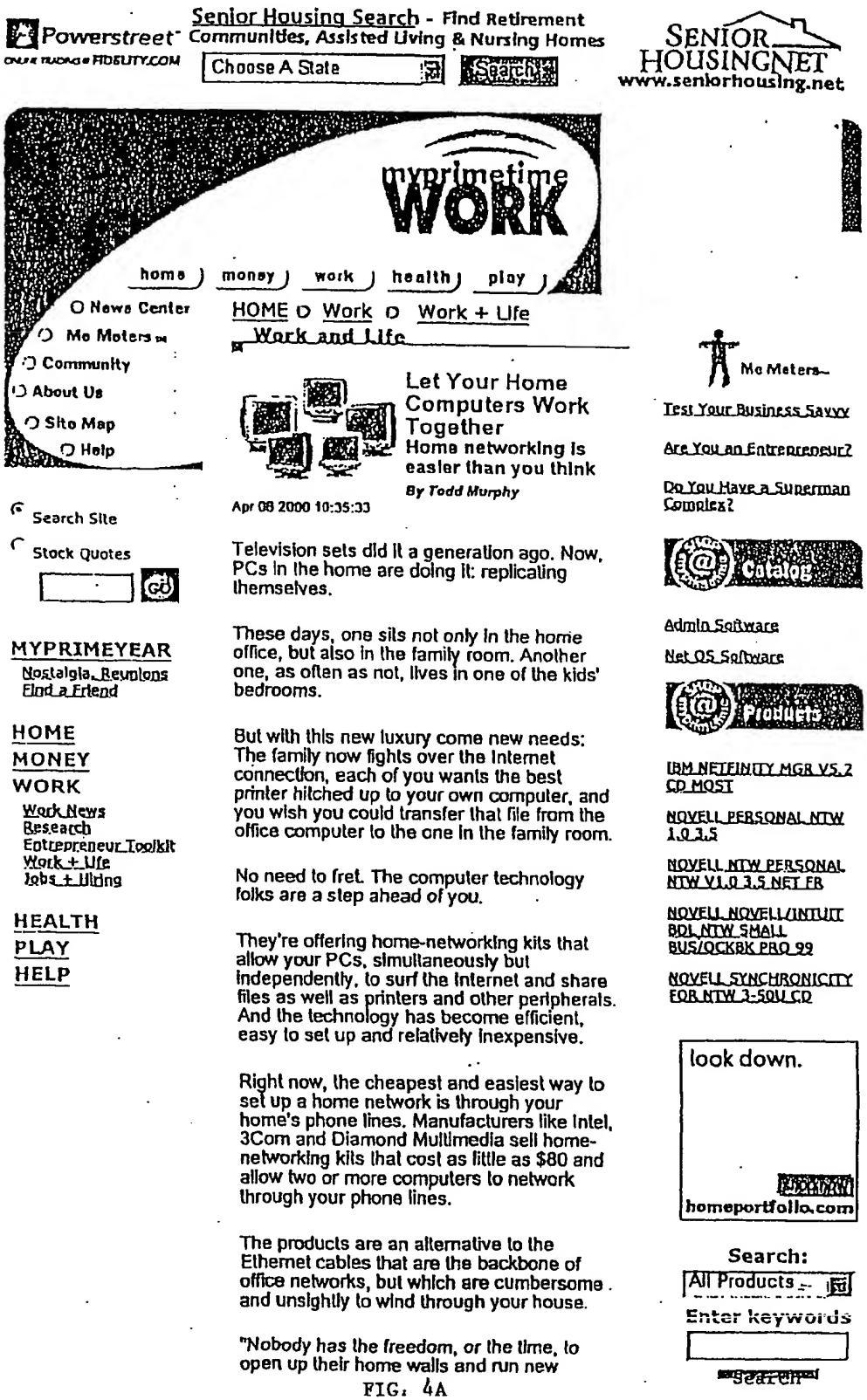


FIG. 4A

Networking your home computers is getting easier and cheaper

cabling," says Mike Reed, director of marketing for Diamond Multimedia, which makes HomeFree Phoneline, one of the leading home-networking products.

The phone line products generally consist of PCI network cards that are installed in the PCs, phone cables and the software that enables it all to work.

You can talk on the phone line while the computers are networking, since the networking equipment uses a different frequency.

Manufacturers generally sell two versions of phone line networking kits — those that transfer data at up to one megabits per second and those that transfer data at up to 10 megabits per second. The newer 10 mbps versions may be over-promoting themselves — some independent tests have shown they work at about half that speed.

Still, even one megabit per second is more than fast enough to quickly download files and print documents. And unless you're surfing the Internet at blistering speeds (remember, even DSL lines generally run at 256 mbps or so) you'll notice no difference in surfing speed, either.

· The Best Home-Networking Kits...

· Want to Go Totally Wireless?

Related Items:
Surf the Net and Receive Phone Calls
Reduce Office Clutter With a Scanner
Guide to Data Recovery Services
Summary of Personal Digital Assistants

Feedback:
E-mail Todd Murphy
Join the wired home office discussion

Web links:
Diamond Multimedia
Intel's Anypoint kit
3com
Webgear.com

Related books:
Networking Home PCs For Dummies, Alan R Neibauer, Jr.
Practical Microsoft Windows Peer Networking, Jerry Lee Ford, Jr.
This Wired Home: The Microsoft Guide to Home Networking, Alan R. Neibauer

Email This Article

Home | Money | Work | Health | Play
E-mail Me | Stock Quotes | "Me" Meters | Message Board | Search | Site Map
About Us | Employment | Privacy Policy | Terms & Conditions | Contact Us
Money sponsor: Fidelity Investments

FIG 4B

Networking your home computers is getting easier and cheaper

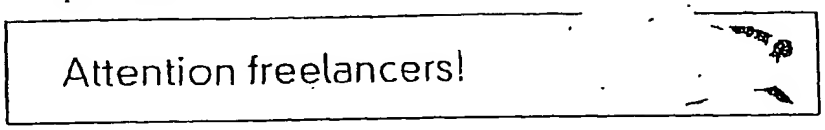


FIG 4C

PERSONAL NTW 1.0.3.5

Media type	Disk
Minimum processor required	286
Minimum RAM requirements	28KB
Warranty	90 days
Minimum graphics required	VGA
CD-ROM supported	No
Sound cards supported	No
Fax boards supported	No
Modems supported	No
Mouse supported	Supported
Tablets supported	No
Scanners supported	No
Optical supported	No
CD-recordable supported	No
Image acceleration supported	No

Warranty

Back to Top

All items sold by Egghead are covered by the manufacturer's original warranty. Click here to view Manufacturer Warranty and Technical Support information.

Sales Policies

Back to Top

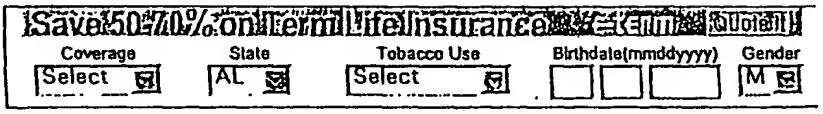
egghead accepts VISA, MasterCard, Discover, American Express, or Optima credit cards for all purchases.

Click Here to view egghead's Payment and Sales policies.

Shipping & Handling

Back to Top

No matter how many products you buy now, Egghead.com Superstores offers free shipping on UPS Ground deliveries. Additional charges apply for expedited shipping.



91684 - 1724 Last Updated: Thursday April 6, 2000 12:58:47 am PDT

FIG 5B